# Simple P Systems with Prescribed Teams of Sets of Rules

Artiom Alhazov[1][0000-0002-6184-3971], Rudolf Freund[2][0000-0003-1255-1953], and Sergiu Ivanov[3][0000-0002-6286-0917]

[1] State University of Moldova,
Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[3] IBISC, Univ. Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

**Abstract.** In this paper we consider simple P systems with prescribed teams of sets of rules, with the application of the rule sets in the teams possibly depending on some given condition, as well as, in the general case, the different sets of rules in a prescribed team working in different derivation modes, whereas in homogeneous systems for all sets of rules the same derivation mode comes into action.

We prove some general results, for example, (i) how with such simple P systems with prescribed teams of sets of rules we can simulate label controlled P systems, where only rules with the same label can be applied, (ii) how simple purely catalytic P systems can be mimicked by simple P systems with prescribed teams of sets of non-cooperative rules with all sets of rules working in the sequential derivation mode, and (iii) how simple catalytic P systems can be mimicked by simple P systems with prescribed teams of sets of non-cooperative rules with some sets of rules working in the sequential derivation mode and only one working in the maximally parallel derivation mode.

Computational completeness of these simple P systems with prescribed teams of sets of non-cooperative rules therefore immediately follows from the well-known results for simple catalytic and purely catalytic P systems, respectively. On the other hand, homogeneous simple P systems with prescribed teams of sets of non-cooperative rules with all teams working in the maximally parallel derivation mode have the same computational power as $ET0L$ systems used for multisets.

Moreover, we also establish specific results, especially for computational completeness, when using other maximally parallel derivation modes than the standard maximally parallel derivation mode, e.g., when using the additional feature of taking the maximal number of rules or the maximal number of objects which are affected or generated.

**Keywords:** applicability condition, computational completeness, derivation modes, $ET0L$ systems, P systems, prescribed teams

## 1 Introduction

A quarter of a century ago, membrane (P) systems were introduced in [12] as a multiset-rewriting model of computing inspired by the hierarchical membrane structure and the functioning of the living cell, with the molecules/objects evolving in parallel. Since then, this area of biologically motivated computing has emerged in a fascinating way. A lot of interesting theoretical models have been developed by scientists all over the world, many of them already documented in two textbooks, see [13] and [14]. For actual information, we refer to the *P systems webpage* [16] as well as to the issues of the *Bulletin of the International Membrane Computing Society* and of the *Journal of Membrane Computing*.

P systems traditionally operate on multisets of objects, hence, the power of non-cooperative rules (even) when working in the maximally parallel derivation mode is rather restricted; for example, the multiset language $\{b^{2^n} \mid n \in \mathbb{N}\}$ cannot be obtained with non-cooperative rules by halting computations. Therefore, one of the fundamental questions which has attracted a lot of attention in the area of P systems is, how variants of different ways of cooperation of the rules and various control mechanisms affect the computational power. For example, allowing for cooperative rules rather easily boosts the power of specific variants of P systems to computational completeness.

One of the well-known control mechanisms forcing some rules to only be applied together (in the sequential derivation mode) are matrix grammars, in which the rules are grouped into sequences, which in the given order must be applied one after another. A less strict variant where the rules in a set of rules called *prescribed teams* can be applied in any order was introduced in [6]. In [4], such prescribed teams are working on different objects.

In contrast to the original model, in which the rules of a team can be applied together only sequentially, we here consider a *team* as a set of sets of rules, where each set of rules has assigned (i) its own applicability condition and (ii) its own derivation mode in which the rules in this set have to be applied, and based on one of these teams a suitable multiset of rules to be applied to the underlying configuration is constructed.

In the model of (*simple*, i.e., only one membrane region is considered) P systems with prescribed teams of sets of rules, the application of a team means applying each set of rules in the chosen team to be used in the derivation mode assigned to the set in this team, provided the applicability condition based on the features of the underlying configuration is fulfilled. In *internally homogenous* systems, all sets of rules in a team have assigned the same derivation mode, whereas in *globally homogenous* systems all teams have assigned the same derivation mode for all sets of rules in the teams. In this paper, we mainly focus on the sequential and the maximally parallel derivation mode; yet we also

establish some results for other derivation modes, as, for example considered in the formal framework for static P systems, see [9], or then defined in [5, 2, 3, 1].

One obvious result we are going to prove is that globally homogenous simple P systems working in the maximally parallel derivation mode for the teams of sets of non-cooperative rules have the same computational power as $ET0L$ systems, i.e., extended tabled Lindenmayer systems. Simple P systems with prescribed teams of sets of rules can simulate label controlled P systems, where only rules with the same label can be applied. Moreover, simple purely catalytic P systems can be mimicked by simple P systems with prescribed teams of sets of non-cooperative rules with the sets of rules working in the sequential derivation mode. For the simulation of catalytic P systems, one additional set working in the maximally parallel derivation mode is needed. Computational completeness of these simple P systems with prescribed teams of sets of non-cooperative rules therefore can immediately be inferred from the well-known results for simple catalytic and purely catalytic P systems, respectively. Furthermore, using sets of symbols as permitting and forbidden context conditions for the sets of rules in the teams allows for an easy direct simulation of register machines, either with using non-cooperative rules or else insertion and deletion rules.

## 2 Definitions

For an alphabet $V$, a finite non-empty set of abstract symbols, the free monoid generated by $V$ under the operation of concatenation, i.e., the set containing all possible strings over $V$, is denoted by $V^*$. The empty string is denoted by $\lambda$, and $V^*\backslash\{\lambda\}$ is denoted by $V^+$. For an arbitrary alphabet $V = \{a_1, \ldots, a_n\}$, the number of occurrences of a symbol $a_i$ in a string $x$ is denoted by $|x|_{a_i}$, while the length of a string $x$ is denoted by $|x| = \sum_{a_i \in V} |x|_{a_i}$. The Parikh vector associated with $x$ with respect to $a_1, \ldots, a_n$ is $(|x|_{a_1}, \ldots, |x|_{a_n})$. The Parikh image of an arbitrary language $L$ over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in $L$, and is denoted by $Ps(L)$. For a family of languages $FL$, the family of Parikh images of languages in $FL$ is denoted by $PsFL$, while for families of languages over a one-letter ($d$-letter) alphabet, the corresponding sets of non-negative integers ($d$-vectors with non-negative components) are denoted by $NFL$ ( $N^dFL$ ).

A (finite) multiset over an alphabet $V = \{a_1, \ldots, a_n\}$, is a mapping $f : V \to \mathbb{N}$ and can be represented by $\langle a_1^{f(a_1)}, \ldots, a_n^{f(a_n)} \rangle$ or by any string $x$ for which $(|x|_{a_1}, \ldots, |x|_{a_n}) = (f(a_1), \ldots, f(a_n))$. In the following we will not distinguish between a vector $(m_1, \ldots, m_n)$, a multiset $\langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ or a string $x$ having $(|x|_{a_1}, \ldots, |x|_{a_n}) = (m_1, \ldots, m_n)$. Fixing the sequence of symbols $a_1, \ldots, a_n$ in an alphabet $V$ in advance, the representation of the multiset $\langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ by the string $a_1^{m_1} \ldots a_n^{m_n}$ is unique. The set of all finite multisets over an alphabet $V$ is denoted by $V^\circ$. The cardinality of a set or multiset $M$ is denoted by $|M|$.

The family of regular, context-free, and recursively enumerable string languages is denoted by $\mathcal{L}(REG)$, $\mathcal{L}(CF)$, and $\mathcal{L}(RE)$, respectively. As $Ps\mathcal{L}(REG) = Ps\mathcal{L}(CF)$, in the area of multiset rewriting $\mathcal{L}(CF)$ plays no role at

all, and in the area of membrane computing we often only get characterizations of $Ps\mathcal{L}(REG)$ and $Ps\mathcal{L}(RE)$ or else $Ps\mathcal{L}(ET0L)$, where $\mathcal{L}(ET0L)$ denotes the family of languages generated by extended tabled Lindenmayer systems ($ET0L$ systems).

For further notions and results in formal language theory we refer to textbooks like [7] and [15].

### 2.1 Register Machines

Register machines are well-known universal devices for computing on (or generating or accepting) sets of vectors of natural numbers. The following definitions and propositions are given as in [1].

**Definition 1.** *A* register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

*where*

- $m$ *is the number of registers,*
- $P$ *is the set of instructions bijectively labeled by elements of* $B$*,*
- $l_0 \in B$ *is the initial label, and*
- $l_h \in B$ *is the final label.*

*The instructions of* $M$ *can be of the following forms:*

- $p : (ADD(r), q, s)$*;* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
  *Increase the value of register* $r$ *by one, and non-deterministically jump to instruction* $q$ *or* $s$*.*
- $p : (SUB(r), q, s)$*;* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
  *If the value of register* $r$ *is not zero then decrease the value of register* $r$ *by one (*decrement *case) and jump to instruction* $q$*, otherwise jump to instruction* $s$ *(*zero-test *case).*
- $l_h : HALT$*.*
  *Stop the execution of the register machine.*

*A* configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.* $M$ *is called* deterministic *if the ADD-instructions all are of the form* $p : (ADD(r), q)$*.*

*Throughout the paper,* $B_{ADD}$ *denotes the set of labels of ADD-instructions* $p : (ADD(r), q, s)$ *of arbitrary registers* $r$*, and* $B_{SUB(r)}$ *denotes the set of labels of all SUB-instructions* $p : (SUB(r), q, s)$ *of a decrementable register* $r$*. Moreover, for any* $p \in B \setminus \{l_h\}$*,* $Reg(p)$ *denotes the register affected by the ADD- or SUB-instruction labeled by* $p$*; for the sake of completeness, in addition* $Reg(l_h) = 1$ *is taken.*

In the *accepting* case, a computation starts with the input of an $l$-vector of natural numbers in its first $l$ registers and by executing the first instruction of $P$ (labeled by $l_0$); it terminates with reaching the $HALT$-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of $P$ (labeled by $l_0$); it terminates with reaching the $HALT$-instruction and the output of a $k$-vector of natural numbers in its last $k$ registers. Without loss of generality, we may assume all registers except the last $k$ output registers to be empty at the end of the computation.

In the *computing* case, a computation starts with the input of an $l$-vector of natural numbers in its first $l$ registers and by executing the first instruction of $P$ (labeled by $l_0$); it terminates with reaching the $HALT$-instruction and the output of a $k$-vector of natural numbers in its last $k$ registers. Without loss of generality, we may assume all registers except the last $k$ output registers to be empty at the end of the computation.

For useful results on the computational power of register machines, we refer to [11]; for example, to prove our main theorem, we need the following formulation of results for register machines generating or accepting recursively enumerable sets of vectors of natural numbers with $k$ components or computing partial recursive relations on vectors of natural numbers:

**Proposition 1.** *Deterministic register machines can accept any recursively enumerable set of vectors of natural numbers with $l$ components using precisely $l+2$ registers. Without loss of generality, we may assume that at the end of an accepting computation all registers are empty.*

**Proposition 2.** *Register machines can generate any recursively enumerable set of vectors of natural numbers with $k$ components using precisely $k+2$ registers. Without loss of generality, we may assume that at the end of a generating computation the first two registers are empty, and, moreover, on the output registers, i.e., the last $k$ registers, no SUB-instruction is ever used.*

**Proposition 3.** *Register machines can compute any partial recursive relation on vectors of natural numbers with $l$ components as input and vectors of natural numbers with $k$ components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last $k$ registers, no SUB-instruction is ever used.*

In all cases it is essential that the output registers never need to be decremented.

### 2.2 Extended Tabled Lindenmayer Systems

An *extended tabled Lindenmayer system* (an $ET0L$ *system* for short) is a construct

$$G = (V, \Sigma, T_1, \ldots, T_n, A) \text{ where}$$

- $V$ is a set of objects;
- $\Sigma \subseteq V$ is a set of *terminal objects*;
- $T_j$, $1 \leq i \leq n$, called *tables* are finite sets of non-cooperative rules over $V$, i.e., of the form $a \to u$ with $a \in V$ and $u \in V^*$;
- $A \in V^+$ is the axiom.

A computation in the $ET0L$ system $G$ starts with the axiom $A$; then, in each computation step, a table $T_j$ is chosen and the rules in $T_j$ are applied to the current configuration in a parallel way. The language generated by $G$ is the set of all terminal strings in $\Sigma^*$ obtained in that way from the axiom $A$, i.e.,

$$L(G) = \{w \in \Sigma^* \mid A \Longrightarrow^* w\}.$$

$ET0L$ systems can also be considered as computing models working on multisets instead of strings, i.e., the axiom $A$ is the initial multiset and the configurations are multisets on which the non-cooperative rules in the tables work in parallel. In the following, such $ET0L$ systems working on multisets will be denoted as $mET0L$ systems. Obviously, we have $\mathcal{L}(mET0L) = Ps\mathcal{L}(ET0L)$.

*Remark 1.* As a technical detail we mention that many authors require every table to contain at least one rule for every object in $V$. We observe that incomplete tables missing a rule for some $x \in V$ can easily be made complete by adding the unit rules $x \to x$ for all $x \in V$ for which so far no rule is already present in the table.

## 3 Simple P Systems

Taking into account the well-known flattening process, which means that computations in a P system with an arbitrary (static) membrane structure can be simulated in a P system with only one membrane, e.g., see [8], in this paper we only consider simple P systems, i.e., with the simplest membrane structure of only one membrane region:

**Definition 2.** *A* simple P system *is a construct*

$$\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, \delta)$$

*where*

- $V$ *is the alphabet of* objects*;*
- $\mathcal{C} \subseteq V$ *is the alphabet of* catalysts*;*
- $\Sigma \subseteq (V \setminus \mathcal{C})$ *is the alphabet of* terminal objects*;*
- $w \in V^\circ$ *is the multiset of objects initially present in the membrane region;*
- $\mathcal{R}$ *is a finite set of* evolution rules *over* $V$*; these evolution rules are multiset rewriting rules* $u \to v$ *with* $u, v \in V^\circ$*;*
- $\delta$ *is the* derivation mode*.*

A *catalytic rule* is of the form $ca \to cv$, a *non-cooperative* rule is of the form $a \to v$, where $c$ is a catalyst, $a$ is an object from $V \setminus \mathcal{C}$, and $v$ is a string from $(V \setminus \mathcal{C})^*$. A simple P system only using catalytic and non-cooperative rules is called *catalytic*, and it is called *purely catalytic* if only catalytic rules are used. The *type* of a (simple) P system only using non-cooperative rules is abbreviated by *ncoo*, the types of catalytic and purely catalytic P systems are abbreviated by *cat* and *pcat*, respectively.

The multiset in the single membrane region of $\Pi$ constitutes a *configuration* of the P system. The *initial configuration* is given by the initial multiset $w$; in case of accepting or computing P systems the input multiset $w_0$ is assumed to be added to $w$, i.e., the initial configuration then is $ww_0$.

A transition between configurations is governed by the application of the evolution rules, which is done in the given derivation mode $\delta$. The application of a rule $u \to v$ to a multiset $M$ results in subtracting from $M$ the multiset identified by $u$, and then in adding the multiset identified by $v$. Observe that each catalyst can be used (at most) once in every derivation step.

If no catalysts are used, we omit $\mathcal{C}$ and simply write $\Pi = (V, \Sigma, w, \mathcal{R}, \delta)$.

### 3.1 Variants of Derivation Modes

Given a P system $\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, \delta)$, the set of multisets of rules applicable to a configuration $C$ is denoted by $Appl(\Pi, C)$.

The set of all multisets of rules applicable to a given configuration can be restricted by imposing specific conditions, thus yielding the following basic derivation modes (for example, see [9] for formal definitions):

- asynchronous mode (abbreviated *asyn*): at least one rule is applied;
- sequential mode (*sequ*): only one rule is applied;
- maximally parallel mode (*max*): a non-extendable multiset of rules is applied;
- maximally parallel mode with maximal number of rules ($max_{rules}$): a non-extendable multiset of rules of maximal possible cardinality is applied;
- maximally parallel mode with maximal number of objects ($max_{objects}$): a non-extendable multiset of rules affecting as many objects as possible is applied.

If $Appl(\Pi, C)$ is not empty, this set equals the set $Appl(\Pi, C, asyn)$ of multisets of rules applicable in the *asynchronous derivation mode* (abbreviated *asyn*).

In [5], these derivation modes are restricted in such a way that each rule can be applied at most once, thus yielding the set modes *sasyn*, *smax*, $smax_{rules}$, and $smax_{objects}$ (the sequential mode is already a set mode by definition).

In this paper we shall often restrict ourselves to the derivation modes *sequ* and *max*:

The set $Appl(\Pi, C, sequ)$ denotes the set of multisets of rules applicable in the *sequential derivation mode* (abbreviated *sequ*), where in each derivation step exactly one rule is applied.

The standard parallel derivation mode used in P systems is the *maximally parallel derivation mode* (*max* for short), in which only non-extendable multisets of rules can be applied:

$$\begin{aligned} Appl(\Pi, C, max) = \{ & R \in Appl(\Pi, C) \mid \\ & \text{there is no } R' \in Appl(\Pi, C) \\ & \text{such that } R' \supset R\}. \end{aligned}$$

In the maximally parallel mode with maximal number of rules $max_{rules}$, a non-extendable multiset of rules of maximal possible cardinality is applied;

$$\begin{aligned} Appl(\Pi, C, max_{rules}) = \{ & R \in Appl(\Pi, C, asyn) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, asyn) \\ & \text{such that } |R'| > |R|\}. \end{aligned}$$

For some new variants of derivation modes we refer to [2, 3]. We recall some of the definitions:

Given a multiset $R$ of rules in $Appl(\Pi, C)$, we write $C \xrightarrow{R} C'$ if $C'$ is the result of applying $R$ to $C$. The number of objects affected by applying $R$ to $C$ is denoted by $\mathrm{Aff}(C, R)$. The number of objects generated in $C'$ by the right-hand sides of the rules applied to $C$ with the multiset of rules $R$ is denoted by $\mathrm{Gen}(C, R)$. The difference between the number of objects in $C'$ and $C$ is denoted by $\Delta obj(C, R)$. In all cases, the catalysts from $\mathcal{C}$ are taken into account, too.

In the maximally parallel mode with maximal number of objects $max_{objects}$, a non-extendable multiset of rules affecting as many objects as possible is applied; we observe that such multisets of rules are non-extendable anyway.

$$\begin{aligned} Appl(\Pi, C, max_{objects}) = \{ & R \in Appl(\Pi, C, asyn) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, asyn) \\ & \text{such that } \mathrm{Aff}(C, R) < \mathrm{Aff}(C, R')\}. \end{aligned}$$

We now consider some of the new derivation modes defined in [2, 3] which start from the non-extendable multisets of rules applicable to the current configuration $C$ as for the derivation mode *max*, which instead of looking at the number of affected objects taken into account, the number of generated objects and the difference of objects between the derived configuration and the current configuration, respectively.

$max_{GENobjects}max$ a non-extendable multiset of rules $R$ applicable to the current configuration $C$ is only taken if the number of objects generated by the application of the rules in $R$ to the configuration $C$ is maximal with respect to the number of objects generated by the application of the rules in

any other non-extendable multiset of rules $R'$ to the configuration $C$:

$$\begin{aligned} Appl(\Pi, C, max_{GENobjects}max) = \{&R \in Appl(\Pi, C, max) \mid \\ &\text{there is no } R' \in Appl(\Pi, C, max) \\ &\text{such that } \mathrm{Gen}(C, R) < \mathrm{Gen}(C, R')\}. \end{aligned}$$

$max_{\Delta objects}max$ a non-extendable multiset of rules $R$ applicable to the current configuration $C$ is only taken if the difference $\Delta C = |C'| - |C|$ between the number of objects in the configuration $C'$ obtained by the application of $R$ and the number of objects in the underlying configuration $C$ is maximal with respect to the differences in the number of objects obtained by applying any other non-extendable multiset of rules:

$$\begin{aligned} Appl(\Pi, C, max_{\Delta objects}max) = \{&R \in Appl(\Pi, C, max) \mid \\ &\text{there is no } R' \in Appl(\Pi, C, max) \\ &\text{such that } \Delta obj(C, R) < \Delta obj(C, R')\}. \end{aligned}$$

### 3.2 Computations in a P System

The P system continues with applying multisets of rules according to the given derivation mode until there remain no applicable rules in the single region of $\Pi$, i.e., as usual, with all these variants of derivation modes as defined above, we consider *halting computations*.

We may generate or accept or even compute functions or relations. The inputs/outputs may be multisets or strings, defined in the well-known way. When the system *halts*, in case of computing with multisets we consider the number of objects from $\Sigma$ contained in the membrane region at the moment when the system halts as the *result* of the underlying computation of $\Pi$.

We would like to emphasize that as results we only take the objects from the terminal alphabet $\Sigma$, especially the catalysts are not counted to the result of a computation. On the other hand, with all the proofs given in this paper, except for the catalysts – if any – no other "garbage" remains in the membrane region at the end of a halting computation, i.e., we could even omit $\Sigma$.

### 3.3 (Simple) P Systems With Label Control

We may extend the model of a simple P system to the model of a *simple P system with label control*

$$\Pi = (V, \mathcal{C}, \Sigma, w, B, \mathcal{R}, \delta)$$

by labelling each rule in $\mathcal{R}$ by an element from a set of labels $B$. Then in any derivation step only rules labeled by the same label $r \in B$ are allowed to be used together. Such controlled P systems were investigated in [10].

*Example 1.* Consider the simple P system of type *ncoo* without catalysts

$$\Pi = (V = \{a, b\}, \Sigma = \{a\}, w = b, B = \{1, 2\}, \mathcal{R} = \{1 : b \to bb, 2 : b \to a\}, max)$$

with the two labels 1 and 2 in $B$ as well as the labeled rules $1 : b \to bb$ and $2 : b \to a$ in $\mathcal{R}$.

Applying rule $1 : b \to bb$ $n \geq 0$ times we obtain $b^{2^n}$; by applying the second rule $2 : b \to a$ we finally obtain the terminal multiset $a^{2^n}$. Hence, $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$, a multiset language which cannot be obtained by a simple P system of type *ncoo* without additional control mechanism.

## 4 Simple P Systems with Prescribed Teams of Sets of Rules

We now consider a new model of simple P systems, where in one derivation step specific sets of rules – called *teams* – are applied in their assigned derivation mode. As usual, we start with a finite multiset of objects until no such team can be applied any more.

**Definition 3.** *A* simple P system with prescribed teams of sets of rules – *a* PPT system *for short – is a construct*

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, A) \quad \text{where}$$

- $V$ *is a set of objects;*
- $\Sigma \subseteq V$ *is a set of* terminal objects*;*
- $P$ *is a finite set of* multiset rules*, i.e., each rule is the form* $u \to v$ *with* $u \in V^*$ *and* $v \in V^+$*;*
- *each* prescribed team $T_j$*,* $1 \leq i \leq n$*, is a finite set of sets of rules from* $P$ *together with the associated derivation mode* $\delta_j$ *and possibly some applicability condition* $K_j$*, i.e.,* $T_j = (\{(K_{j,i}, R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\})$*, where the* $R_{j,i} \subseteq P$ *are finite sets of rules from* $P$*;*
- $A \in V^\circ$ *is a finite multiset of initial objects from* $V$*.*

As usual, a rule $p \in P$, $p = u \to v$, is called *applicable* to a *configuration*, i.e., an object $x \in V^\circ$, if and only if $u$ is a subset of $x$. The set of all rules applicable to $x$ is denoted by $Appl(\Pi, x)$

The number $n$ of teams is called the *degree* of $\Pi$. $|T_j|$ is called the *size* of the prescribed team $T_j$. If all prescribed teams have at most size $s$, then $\Pi$ is called a *PPT system of size* $s$. If the number of rules in the sets of rules is at most $m$, then $\Pi$ is called a *PPT system of rule size* $m$. $\Pi$ is called a *PPT system of type* $(n, s, m)$, if it is of degree $n$, size $s$, and rule size $m$. Moreover, if all rules in the sets of rules in the teams are of a specific type $\alpha$ (for example *ncoo*), we call $\Pi$ a *PPT system of type* $(\alpha; n, s, m)$.

The family of sets of multisets generated/accepted by PPT system of type $(\alpha; n, s, m)$ is denoted by $\mathcal{L}(PPT_{gen}(\alpha; n, s, m))/\mathcal{L}(PPT_{acc}(\alpha; n, s, m))$. Any of

the parameters $n, s, m$ can be replaced by $*$, if the number cannot be bounded; $\alpha$ can also be omitted.

As derivation modes, we will often restrict ourselves to the sequential derivation mode *sequ* and the maximally parallel derivation mode $max$. In addition, in some cases we will also consider the derivation modes $max_{rules}$, $max_{objects}$ and $max_{GENobjects}max$, $max_{\Delta objects}max$ as well as the set derivation modes $sasyn$, $smax$, $smax_{rules}$, $smax_{objects}$.

The conditions $K_{j,i}$ in the most general case can be any computable/recursive features of the underlying configuration. Here we essentially will consider random context conditions, i.e., $K_{j,i} = (P_{j,i}, Q_{,j,i})$, where $P_{j,i}$ and $Q_{j,i}$ are finite sets of multisets over $V$; $P_{j,i}$ is the set of *permitting contexts* and $Q_{j,i}$ is the set of *forbidden contexts*. The random context condition $K_{j,i} = (P_{j,i}, Q_{j,i})$ is fulfilled by a multiset $x$ if and only if $x$ contains each multiset in $P_{j,i}$, but none of the multisets in $Q_{j,i}$. If no conditions $K_{j,i}$ are specified, we simply write $T_j = \{(R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\}$.

If different derivation modes appear in a team, the whole PPT system is called *non-homogenous*. The system is called *locally homogenous*, if for all teams, the sets of rules in the team all are applied in the same derivation mode $\delta_j$, and we write $T_j = (\{R_{j,i} \mid 1 \leq i \leq n_j\}, \delta_j)$, $1 \leq j \leq n$. Finally, the system is called *globally homogenous* if the derivation mode is the same $\delta$ for all $T_j$, and we only write $T_j = \{R_{j,i} \mid 1 \leq i \leq n_j\}$ and specify $\delta$ by writing $\Pi = (V, \Sigma, P, T_1, \ldots, T_n, \delta, A)$.

### 4.1 Computations in a PPT system

Given a prescribed team of sets of rules

$$T_j = (\{(K_{j,i}, R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\})$$

with the derivation modes

$$\begin{aligned}\{\delta_{j,i} \mid 1 \leq i \leq n_j\} \subseteq\ & \{sequ, max, asyn\} \\ & \cup \{max_{rules}, max_{objects}\} \\ & \cup \{smax_{GENobjects}max, smax_{\Delta objects}max\} \\ & \cup \{max_{GENobjects}max, max_{\Delta objects}max\} \\ & \cup \{sasyn, smax, smax_{rules}, smax_{objects}\}\end{aligned}$$

a derivation step with $T_j$ on the configuration $x$ can be carried out in the following way:

1. we choose a multiset of rules $R \in Appl(\Pi, x)$; the multiset of objects which the rules in $R$ bind is denoted by $Bind(R, x)$, the multiset of objects in $x \setminus Bind(R, x)$ is denoted by $Idle(R, x)$;
2. we now for all $1 \leq i \leq n_j$ check whether $x$ fulfills the applicability conditions $K_{j,i}$;

3. each rule in $R$ must be assigned to one of the sets $R_{j,i}$ for which the applicability condition $K_{j,i}$ is fulfilled, yielding the multiset of rules $R'_{j,i}$; the multiset of objects which the rules in $R'_{j,i}$ bind is denoted by $Bind(R, R'_{j,i}, K)$;
4. for $\delta_{j,i} = asyn$ we only have to check if either $|R'_{j,i}| > 0$ or, if $|R'_{j,i}| = 0$, then we have to check that no rule from $R_{j,i}$ can be applied to objects from $Idle(R, x)$; otherwise the check fails, i.e., $R$ cannot be applied; observe that, due to the competition for objects by the different sets of rules in the team we do not require a rule to be applied for at least one a rule in $R_{j,i}$;
5. for $\delta_{j,i} = sasyn$ we only have to check if either $|R'_{j,i}| \geq 1$ where each rule in $R'_{j,i}$ is applied only once, or $|R'_{j,i}| = 0$, in which case we have to check that no rule from $R_{j,i}$ can be applied to objects from $Idle(R, x)$; otherwise the check fails, i.e., $R$ cannot be applied;
6. for $\delta_{j,i} = sequ$ we first check whether $|R'_{j,i}| = 1$; if $|R'_{j,i}| = 0$, then we have to check that no rule from $R_{j,i}$ can be applied to objects from $Idle(R, x)$; if $|R'_{j,i}| > 1$ the check fails, i.e., $R$ cannot be applied; again we emphasize that we do not require a rule to be applied for one a rule in $R_{j,i}$;
7. for any $\delta_{j,i}$ being one of the maximally parallel derivation modes $max$, $max_{rules}$, $max_{objects}$ as well as $max_{GENobjects}max$, $max_{\Delta objects}max$ we now check that $R'_{j,i}$ fulfills the conditions required by the derivation mode on the subconfiguration given by the objects in $Bind(R, R'_{j,i}, K) \cup Idle(R, x)$;
8. for any $\delta_{j,i}$ being one of the set derivation modes $smax$, $smax_{rules}$, $smax_{objects}$ as well as $smax_{GENobjects}max$, $smax_{\Delta objects}max$ we now check that $R'_{j,i}$ fulfills the conditions required by the derivation mode on the subconfiguration given by the objects in $Bind(R, R'_{j,i}, K) \cup Idle(R, x)$;
9. if all checks from above have been passed correctly, the multiset of rules $R$ can be applied to the current configuration $x$.

We emphasize that the rule sets in a team compete for the objects available in the underlying configuration, but at the end each set of rules for itself makes its part of the transition from the underlying configuration to the next configuration in a correct way according to its assigned derivation mode, as no additional rule could bind an idle object. Moreover, we observe that a set of rules $R_{j,i}$ from $T_j$ can only be chosen if the applicability condition $K_{j,i}$ is fulfilled by $x$. Finally, a team $T_j$ can only be applied if the multiset of rules obtained by the procedure described above is not empty.

As some variant of the general model we may also consider prescribed teams of sets of rules for which the applicability conditions $K_{j,i}$ are the same for all $1 \leq i \leq n_j$, i.e., just one condition $K_j$, and then we write

$$T_j = (K_j, \{(R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\})$$

and can simplify the procedure for applying $T_j$ by first checking that the current configuration fulfills $K_j$.

As a first example we show how label control can easily be simulated by teams of sets of rules:

*Example 2.* Consider the globally homogenous PPT system of type *ncoo*

$$\Pi = (V = \{a, b\}, \Sigma = \{a\}, P, T_1, T_2, max, b) \text{ where}$$

$P = \{b \to bb, b \to a\}$, $T_1 = \{\{b \to bb\}\}$, and $T_2 = \{\{b \to a\}\}$. $\Pi$ is a globally homogenous PPT system of type $(ncoo; 2, 1, 1)$.

Applying team $T_1$, i.e., the rule $b \to bb$, in the maximally parallel way $n \geq 0$ times we obtain $b^{2^n}$; by applying the second team $T_2$, i.e., the rule $b \to a$, in the maximally parallel way once, we finally obtain the terminal multiset $a^{2^n}$ as in Example 1. Hence, we conclude $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$ as well as

$$\{a^{2^n} \mid n \geq 0\} \in \mathcal{L}(gh(max)PPT_{gen}(ncoo; 2, 1, 1)),$$

with the prefix $gh(max)$ indicating that we consider globally homogenous PPT systems working in the derivation mode $max$.

## 5 PPT Systems Simulating P Systems With Label Control

As can already be guessed by looking at Example 2, PPT systems can easily simulate P systems with label control – *without catalysts* – which are working in the derivation mode $max$ by putting the rules with the same labels into one team:

**Theorem 1.** *Every P systems with label control $\Pi$, without catalysts, and working in the derivation mode max, can be simulated by a PPT system of type $(n, 1, *)$, where $n$ is the number of different labels for the rules in $\Pi$.*

*Proof.* Given a simple P system with label control, without catalysts,

$$\Pi = (V, \Sigma, w, B, \mathcal{R}, max)$$

where each rule in $\mathcal{R}$ is labelled by an element from a set of labels $B$, $B = \{l_j \mid 1 \leq j \leq n\}$, we construct a globally homogenous PPT system $\Psi$ of degree $n$ and size 1, simulating (the computations of) $\Pi$:

$$\Psi = (V, \Sigma, P, T_1, \ldots, T_n, max, w)$$

where we define

$$P = \{p \mid l_j : p \in \mathcal{R},\ 1 \leq j \leq n\}$$

as well as the teams $T_j$, $1 \leq j \leq n$, as follows:

$$T_j = \{\{p \mid l_j : p \in \mathcal{R}\}\}$$

By definition, the size of $T_j$ is 1, whereas the number of rules in the single set of rules in a team can be arbitrarily large.

We observe that applying the set of rules in in the team $T_j$ in $\Psi$ in the maximally parallel way has the same effect as applying exactly the rules with label $l_j$ in $\Pi$ in the maximally parallel way. □

## 6 PPT Systems Simulating $mET0L$ Systems

We now show that the computational power of $mET0L$ systems equals the computational power of globally homogenous PPT systems of type *ncoo* without applicability conditions working in the derivation mode $max$.

**Theorem 2.** *Every $mET0L$ system with $n$ tables can be simulated by a globally homogenous PPT system of type ncoo, degree $n$, and size 1 without applicability conditions working in the derivation mode $max$.*

*Proof.* The $mET0L$ system $G = (V, \Sigma, T'_1, \ldots, T'_n, A)$ can be simulated by the globally homogenous PPT system of type *ncoo*, degree $n$, and size 1 without applicability conditions working in the derivation mode $max$

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, max, A)$$

where we simply take

$$T_j = \{T'_j \setminus \{a \to a \mid a \in \Sigma\}\}, \quad 1 \leq j \leq n,$$

i.e., the work of the table $T'_j$ is simulated by the single set of rules in the team $T_j$ of the PPT system $\Pi$.

We observe that we have to exclude the unit rules $a \to a$ for the terminal symbols $a \in \Sigma$, from the sets of rules $T'_j$, $1 \leq j \leq n$, in order to ensure that $\Pi$ halts as soon as a terminal configuration (i.e., a configuration only containing terminal symbols) has been reached. Finally, we mention that every (useless) team $T_j$ of the form $\{\emptyset\}$ is to be omitted. □

In order to show the inverse inclusion, we need the following lemma:

**Lemma 1.** *Every globally homogenous PPT system of degree $n$ and size $k$ without applicability conditions working in the derivation mode $max$ can be simulated by a globally homogenous PPT system of degree $n$ and size 1 without applicability conditions working in the derivation mode $max$.*

*Proof.* The globally homogenous PPT system of degree $n$ and size $k$ without applicability conditions working in the derivation mode $max$

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, max, A)$$

with $T_j = \{R_{j,i} \mid 1 \leq j \leq n_j\}$, $1 \leq j \leq n$, can be simulated by the corresponding globally homogenous PPT system of degree $n$ and only size 1 without applicability conditions working in the derivation mode $max$

$$\Pi' = (V, \Sigma, P, T'_1, \ldots, T'_n, max, A)$$

where we take $T'_j = \{\{y \mid y \in R_{j,i},\ 1 \leq i \leq n_j\}\}$. We observe that by definition the rules in the $R_{j,i}$ work in parallel on the underlying configurations in the same way if they are grouped in the $R_{j,i}$ or just in one set of rules $\{y \mid y \in R_{j,i},\ 1 \leq i \leq n_j\}$. We observe that $\Pi'$ again is of degree $n$, but only of size 1. □

Based on this lemma, we now can show how a globally homogenous PPT system of type *ncoo*, degree $n$ without applicability conditions working in the derivation mode *max* can be simulated by an $mET0L$ system with $n$ tables:

**Theorem 3.** *Every globally homogenous PPT system of type ncoo, degree $n$, and size $k$ without applicability conditions working in the derivation mode max can be simulated by an $mET0L$ system with $n$ tables.*

*Proof.* According to Lemma 1, without loss of generality we may assume that the size $k$ is only one. Hence, we may start with a globally homogenous PPT system of type *ncoo*, degree $n$, and size 1 without applicability conditions working in the derivation mode *max*

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, max, A)$$

where for the teams $T_j$ we have $T_j = \{T'_j\}$, $1 \leq j \leq n$, with $T'_j$ being a set of non-cooperative rules.

Then the $mET0L$ system

$$G = (V, \Sigma, T'_1, \ldots, T'_n, A)$$

simulates the (computations of the) PPT system $\Pi$, as the work of the table $T'_j$ simulates the application of the team $T_j$ of the PPT system $\Pi$ with the single set of rules $T'_j$.

As a technical detail we mention that the tables $T'_j$ have to be extended by unit rules $x \to x$ for every $x \in V$ for which no rule is already present in it, in order to fulfill the requirement for $ET0L$ systems as already discussed in Remark 1. □

In sum, we have shown the following result (where $gh(max)PPT(ncoo)$ denotes the globally homogenous PPT systems of type *ncoo* working in the derivation mode *max*):

**Theorem 4.** $\mathcal{L}(mET0L) = \mathcal{L}(gh(max)PPT(ncoo))$.

## 7 PPT Systems Simulating [Purely] Catalytic P Systems

We first consider purely catalytic P systems, which correspond to PPT systems where all sets of non-cooperative rules in the unique team work in the sequential derivation mode.

**Theorem 5.** *Every purely catalytic P system with $n$ catalysts can be simulated by a corresponding globally homogenous PPT system of type ncoo, degree 1, and size $n$ without applicability conditions working in the derivation mode sequ.*

*Proof.* The purely catalytic P system with $n$ catalysts

$$\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, max),$$

i.e., $\mathcal{C} = \{c_k \mid 1 \leq k \leq n\}$, can be simulated by the globally homogenous PPT system of type *ncoo*, degree 1, and size $n$ without applicability conditions working in the sequential derivation mode

$$\Pi = (V, \Sigma, P, T_1, sequ, w)$$

with $T_1 = \{R_{1,k} \mid 1 \leq k \leq n\}$ and

$$P = \{a \to u \mid c_k a \to c_k u \in \mathcal{R} \text{ for some } 1 \leq \mathrm{k} \leq \mathrm{n}\}$$

as well as

$$R_{1,k} = \{a \to u \mid c_k a \to c_k u \in \mathcal{R}\},\ 1 \leq k \leq n.$$

The applicability of the unique team works by applying (at most) one rule $a \to u$ from each $R_{1,k}$, $1 \leq k \leq n$, which corresponds to applying the corresponding rule $c_k a \to c_k u$ in $\Pi$. □

Simple catalytic P systems with $n$ catalysts can be mimicked by simple P systems with prescribed teams of sets of rules, where as in the case of purely catalytic P systems the work of the $n$ catalysts is simulated by $n$ sets of non-cooperative rules in the team working in the sequential mode and one additional set of non-cooperative rules simulates the set of non-catalytic rules working in the maximally parallel derivation mode.

**Theorem 6.** *Every catalytic P system with $n$ catalysts can be simulated by a corresponding PPT system of type ncoo, degree 1, and size $n+1$ without applicability conditions with $n$ components of the unique team working in the derivation mode sequ and one working in the derivation mode max.*

*Proof.* The catalytic P system with $n$ catalysts

$$\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, max),$$

i.e., $\mathcal{C} = \{c_k \mid 1 \leq k \leq n\}$, can be simulated by the PPT system of type *ncoo*, degree 1, and size $n + 1$ without applicability conditions

$$\Pi = (V, \Sigma, P, T_1, w)$$

with $T_1 = \{R_{1,k} \mid 1 \leq k \leq n + 1\}$ and

$$P = \{a \to u \mid c_k a \to c_k u \in \mathcal{R} \text{ for some } 1 \leq \mathrm{k} \leq \mathrm{n}\} \cup \{\mathrm{a} \to \mathrm{u} \mid \mathrm{a} \to \mathrm{u} \in \mathcal{R}\}$$

as well as

$$R_{1,k} = (\{a \to u \mid c_k a \to c_k u \in \mathcal{R}\}, sequ),\ 1 \leq k \leq n,$$

and

$$R_{1,n+1} = (\{a \to u \mid a \to u \in \mathcal{R}\}, max),$$

The application of the unique team works by applying (at most) one rule $a \to u$ from each $R_{1,k}$, $1 \leq k \leq n$, which corresponds to applying the corresponding rule $c_k a \to c_k u$ in $\Pi$, as well as the rules in $R_{1,n+1}$ in the maximally parallel way.

Observe that in contrast to the globally homogenous PPT systems for simulating purely catalytic P systems, we now have non-homogenous PPT systems, as we have to use both derivation modes *sequ* and *max* in the unique team. □

According to Propositions 2 and 1, from Theorems 5 and 6 we immediately infer the following results:

**Corollary 1.** *For any $d \geq 1$, we have*

1. $N^d\mathcal{L}(RE) = \mathcal{L}(PPT_{gen}(ncoo; 1, 3, *))$ *and*
2. $N^d\mathcal{L}(RE) = \mathcal{L}(PPT_{acc}(ncoo; 1, d+3, *))$*;*

*moreover,*
$Ps\mathcal{L}(RE) = \mathcal{L}(PPT_{gen}(ncoo; 1, *, *)) = \mathcal{L}(PPT_{acc}(ncoo; 1, *, *))$.
*In all cases, the degree of the PPT systems is only* 1*.*

## 8 PPT Systems Directly Simulating Register Machines

In this section we show how register machines can directly be simulated by PPT systems in an easy way when using applicability conditions represented by sets of permitting and forbidden contexts, see the defintion on page 11.

**Theorem 7.** *The computations of a register machine can be simulated by a globally homogenous PPT system of type ncoo and size* 2 *using permitting and forbidden contexts as applicability conditions in the sequential derivation mode.*

*Proof.* Consider the register machine

$$M = (m, B, l_0, l_h, R)$$

with $B_{ADD}$ denoting the set of labels of *ADD*-instructions $p : (ADD(r), q, s)$ of arbitrary registers $r$, and $B_{SUB(r)}$ denoting the set of labels of all *SUB*-instructions $p : (SUB(r), q, s)$ of a decrementable register $r$. Moreover, for any $p \in B \backslash \{l_h\}$, $Reg(p)$ denotes the register affected by the *ADD*- or *SUB*-instruction labeled by $p$.

We now construct the globally homogenous PPT system of size 2 working in the sequential derivation mode using permitting and forbidden contexts as applicability conditions

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, sequ, w).$$

Throughout the computation of $\Pi$, one symbol $p \in B$ represents the instruction from the register machine to be simulated next, and the number of symbols $a_r$ represents the contents of register $r$, $1 \leq r \leq m$. Hence, we start with the axiom $w = l_0 w_0$, where $w_0$ represents the initial contents of the registers. If the final label $l_h$ appears, we know that the computation in $M$ has been successful and finally can erase $l_h$, so that a multiset over $\Sigma$ remains as the result of the computation.

Moreover, the set of symbols $V$ only consists of the labels in $B$ and the symbols $a_r$ representing the registers:

$$V = B \cup \{a_r \mid 1 \leq r \leq m\}.$$

The set of terminal symbols $\Sigma$ consists of only those symbols from the set $\{a_r \mid 1 \leq r \leq m\}$ which represent output registers.

We need the following simple non-cooperative rules in $P$ for the simulation of the instructions of $M$:

$$\begin{aligned} P = &\{p \to q a_r,\ p \to s a_r \mid p : (ADD(r), q, s) \in R\} \\ &\cup \{p \to q,\ p \to s \mid p : (SUB(r), q, s) \in R\} \\ &\cup \{a_r \to \lambda \mid 1 \leq r \leq m \text{ and r is a decrementable register}\} \\ &\cup \{l_h \to \lambda\} \end{aligned}$$

The teams of sets of rules with applicability conditions for simulating the instructions of the register machine defined below form the teams $T_1, \ldots, T_n$.

- $p : (ADD(r), q, s)$, $p \in B_{ADD}$, is simulated by the team

  $R_p = \{((\{p\}, \emptyset), \{p \to q a_r,\ p \to s a_r\})\}$
  which can also be written in a simpler way as
  $R_p = \{\{p \to q a_r, p \to s a_r\}\}$, because the rules in this set of rules in this team can anyway only be applied if $p$ is present.

- $p : (SUB(r), q, s)$; $p \in B_{SUB(r)}$, is simulated by the team of sets of rules

  $R_{p,1} = ((\{p, a_r\}, \emptyset), \{\{p \to q\}, \{a_r \to \lambda\}\})$

  (both the presence of $p$ and $a_r$ have to be checked in order to guarantee that both rules $p \to q$ and $a_r \to \lambda$ are applied) as well as by the team of sets of rules

  $R_{p,2} = \{((\{p\}, \{a_r\}), \{p \to s\})\}$

  (both the presence of $p$ and the absence of $a_r$ have to be checked to guarantee that we only proceed to label $s$ if no symbol $a_r$ is present).

- $l_h : HALT$ is simulated by the team
  $R_h = \{((\{l_h\}, \emptyset), \{l_h \to \lambda\})\}$,
  which can also be written in a simpler way as
  $R_h = \{\{l_h \to \lambda\}\}$.

Only in the case of applying a team $R_{p,1}$ two rules are applied in one step, otherwise only one rule is applied.

The application of a team is only possible if the current label symbol $p$ appears in the underlying configuration, and in the case of a $SUB$-instruction also the presence/absence of $a_{Reg(p)}$ is correctly given. Throughout the computation in $\Pi$ exactly one of the teams of sets of rules is applicable before finally a configuration only containing terminal symbols is reached. □

## 9 Computational Completeness

According to Subsection 2.1, register machines are a model being computationally complete for multisets, i.e., every partial recursive relation on multisets can be computed by a register machine. Hence, from Theorem 7 we immediately infer the following result:

**Theorem 8.** *PPT systems of type ncoo and size* 2 *using applicability conditions represented by sets of permitting and forbidden contexts in the sequential derivation mode are computationally complete for multisets.*

Instead of non-cooperative rules we can also use the simple rules of insertion and deletion:

- $I(a)$ inserts an object $a$ in the underlying multiset (and can be interpreted as the rule $\lambda \to a$).
- $D(a)$ deletes an object $a$ from the underlying multiset, if at least one $a$ is present (and can be interpreted as the rule $a \to \lambda$).

Based on the proof of Theorem 7, we easily get the following result for PPT systems using insertion and deletion rules (called PPT systems of type $InsDel$ for short):

**Corollary 2.** *PPT systems of type $InsDel$ and size* 3 *using applicability conditions represented by sets of permitting and forbidden contexts in the sequential derivation mode are computationally complete for multisets.*

*Proof.* Consider the register machine

$$M = (m, B, l_0, l_h, R)$$

with $B_{ADD}$ denoting the set of labels of $ADD$-instructions $p : (ADD(r), q, s)$ of arbitrary registers $r$, and $B_{SUB(r)}$ denoting the set of labels of all $SUB$-instructions $p : (SUB(r), q, s)$ of a decrementable register $r$. Moreover, for any $p \in B \backslash \{l_h\}$, $Reg(p)$ denotes the register affected by the $ADD$- or $SUB$-instruction labeled by $p$.

We now construct the globally homogenous PPT system of type $InsDel$ and size 3 working in the sequential derivation mode using permitting and forbidden contexts as applicability conditions

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, sequ, w).$$

Throughout the computation of $\Pi$, one symbol $p \in B$ represents the instruction from the register machine to be simulated next, and the number of symbols $a_r$ represents the contents of register $r$, $1 \leq r \leq m$. Hence, we start with the axiom $w = l_0 w_0$, where $w_0$ represents the initial contents of the registers. If the final label $l_h$ appears, we know that the computation in $M$ has been successful and finally can erase $l_h$, so that a multiset over $\Sigma$ remains as the result of the computation.

Moreover, the set of symbols $V$ only consists of the labels in $B$ and the symbols $a_r$ representing the registers:

$$V = B \cup \{a_r \mid 1 \leq r \leq m\}.$$

The set of terminal symbols $\Sigma$ consists of only those symbols from the set $\{a_r \mid 1 \leq r \leq m\}$ which represent output registers.

We need the following simple insertion and deletion rules in $P$ for the simulation of the instructions of $M$:

$$\begin{aligned} P = &\{I(p), D(p) \mid p \in B\} \\ &\cup \{I(a_r) \mid 1 \leq r \leq m\} \\ &\cup \{D(a_r) \mid 1 \leq r \leq m \text{ and r is a decrementable register}\} \end{aligned}$$

The teams of sets of rules with applicability conditions for simulating the instructions of the register machine defined below form the teams $T_1, \ldots, T_n$.

- $p : (ADD(r), q, s)$, $p \in B_{ADD}$, is simulated by the team

$$\begin{aligned} R_p = \{&((\{p\}, \emptyset), \{D(p)\}), \\ &((\{p\}, \emptyset), \{I(q),\ I(s)\}), \\ &((\{p\}, \emptyset), \{I(a_r)\})\}, \end{aligned}$$

  which in a shorter way could be written as

$$R_p = ((\{p\}, \emptyset), \{\{D(p)\}, \{I(q),\ I(s)\}, \{I(a_r)\}\})$$

  as the applicability condition $(\{p\}, \emptyset)$ is required for all sets of rules in the team. Observe that the size of these teams now is 3!

- $p : (SUB(r), q, s)$; $p \in B_{SUB(r)}$, is simulated by the team

$$R_{p,1} = ((\{p, a_r\}, \emptyset), \{\{D(p)\}, \{I(q)\}, \{D(a_r)\}\})$$

  (again the size of these teams now is 3)

  as well as by the team

$$R_{p,2} = ((\{p\}, \{a_r\}), \{\{D(p)\}, \{I(s)\}\})$$

– $l_h : HALT$ is simulated by the team

$$R_h = \{\{D(l_h)\}\}.$$

Throughout the computation in $\Pi$ exactly one of the teams of sets of rules is applicable before finally a configuration only containing terminal symbols is reached. □

As is well-known, catalytic and purely catalytic P systems are computationally complete (for multisets), too. Therefore, based on the results shown in Section 7 we get the following results (compare with Corollary 1):

**Corollary 3.** *Globally homogenous PPT systems of type ncoo and degree* 1 *without applicability conditions working in the sequential derivation mode are computationally complete for multisets.*

**Corollary 4.** *PPT systems of type ncoo and degree* 1 *without applicability conditions with one set of rules in the unique team working in the maximally parallel derivation mode and all the other sets of rules working in the sequential derivation mode are computationally complete for multisets.*

## 10 Conclusion

In this paper we have considered the concept of using prescribed teams of sets of rules being applied in different derivation modes, with the applicability of a team possibly depending on a given condition. Among other general results, we have shown that simple purely catalytic P systems with $n$ catalysts can be simulated by simple P systems with one prescribed team of sets of rules with all $n$ sets of non-cooperative rules in this team working in the sequential derivation mode thus simulating the work of the $n$ catalysts, as well as that simple catalytic P systems with $n$ catalysts can be simulated by simple P systems with one prescribed team of sets of non-cooperative rules, where one set of this team works in the maximally parallel derivation mode and the other $n$ sets of rules in this team work in the sequential mode thus again simulating the work of the $n$ catalysts. From the results known for simple (purely) catalytic P systems, we immediately infer the corresponding computational completeness results for the new variants of simple P systems, with on one hand only using non-cooperative rules and no applicability conditions. On the other hand, we can show computational completeness for different variants of simple P systems with prescribed teams of sets of non-cooperative rules by directly simulating register machines, thereby using applicability conditions given as sets of (atomic) promoters and inhibitors.

Throughout this paper, we have restricted ourselves to the two basic derivation modes, i.e., the sequential one and the maximally parallel derivation mode. A thorough investigation of simple P systems with prescribed teams of sets of

rules using other derivation modes remains for future research. Moreover, other kinds of rules might be used, too; for example, insertion and deletion rules instead of non-cooperative rules as already used for simulating register machines in Corollary 2.

## Acknowledgements

The authors gratefully acknowledge the useful comments of the anonymous referees.

Artiom Alhazov acknowledges project 20.80009.5007.22 "Intelligent information systems for solving ill-structured problems, processing knowledge and big data" by the National Agency for Research and Development.

## References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. When catalytic P systems with one catalyst can be computationally complete. *Journal of Membrane Computing*, 3(3):170–181, 2021.
2. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Marion Oswald. Variants of simple purely catalytic P systems with two catalysts. In György Vaszil, Claudio Zandron, and Gexiang Zhang, editors, *International Conference on Membrane Computing ICMC 2021, Proceedings*, pages 39–53, 2021.
3. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Sergey Verlan. Variants of simple P systems with one catalyst being computationally complete. In György Vaszil, Claudio Zandron, and Gexiang Zhang, editors, *International Conference on Membrane Computing ICMC 2021, Proceedings*, pages 21–38, 2021.
4. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Sergey Verlan. Prescribed teams of rules working on several objects. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality – 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 – September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2022.
5. Artiom Alhazov, Rudolf Freund, and Sergey Verlan. P systems working in maximal variants of the set derivation mode. In Alberto Leporati, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers*, volume 10105 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2017.
6. E. Csuhaj-Varjú, J. Dassow, and J. Kelemen. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Topics in computer mathematics. Gordon and Breach, 1994.
7. Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
8. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Claudio Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.

9. Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) P systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2007.
10. Kamala Krithivasan, Gh. Păun, and Ajeesh Ramanujan. On controlled P systems. *Fundam. Inform.*, 131(3–4):451–464, 2014.
11. Marvin L. Minsky. *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
12. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
13. Gheorghe Păun. *Membrane Computing: An Introduction.* Springer, 2002.
14. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
15. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages.* Springer, 1997.
16. The P Systems Website. `http://ppage.psystems.eu/`.